CS5434 Defending Computer Networks
Final project (**spec revision 2, updated 11/20**)
Due: 11.59pm, Wednesday 12/2

HTTP proxy-based malware detector

For this final project, you will work in groups of two to build a HTTP malware detector. You may switch teams for this project; please use Piazza and CMS to form your groups.

First, download the updated VM from the following URL (it's a huge download, so start early). Do not reuse VMs from any of the earlier assignments. All relevant libraries and software needed for the homework is included in this VM.

http://www.cs.cornell.edu/~zteo/cs5434_final_project.zip

Some skeleton code and a Makefile has been set up for you in the project directory. The password for this VM is "**nukacola**". If your VM does not recognize this password, you have an outdated image and should download the updated VM.

We may update project specifications and include supplementary material. **It is your responsibility to monitor Piazza for any updates and material**.

Important notice

You will be working with malware for this project. **The samples provided for this project are not fakes!** Do not attempt to execute any malware. Your computer (and others) may be toast and we cannot help you or grant you a deadline extension. As a reminder, you can get in **very serious** disciplinary trouble for propagating malware, even if it is a result of an accident within an academic setting.

Objective

The objective of this assignment is to identify web-based malware and provide active intervention. In the process, you will learn about proxies, POSIX threads, the Berkeley sockets API, HTTP, zlib and malware analysis through ClamAV.

Background

In a proxy setup, a browser does not make direct web connections to host sites, but instead connects to a proxy server to request pages on its behalf. The proxy server then makes connections to the host site, retrieves the requested pages, and forwards them back to the browser. By acting as the 'man-in-the-middle', the proxy server can intervene during the resource retrieval process.

The Firefox browser for this VM has been set up to use a proxy server at localhost:8080. Under normal circumstances when the proxy server process isn't online, Firefox browsing will not work. Your task is to implement the proxy server so that a user can surf the web on Firefox without fear of encountering malware.

Generic workflow

A generic workflow for this project is as follows:

1. A URL is entered into Firefox, or a link is clicked in the browser.
2. Firefox makes one or more concurrent connections to your proxy server, requesting for files from the web.
3. For each file requested, your proxy server establishes a connection to the relevant website and retrieves the content.
4. The content may be chunked and/or compressed. You need to recombine/decompress the content as necessary and store the uncompressed data into a file.
5. The file is passed through the ClamAV malware detector to check for dangerous content.
6. If the file is safe, the contents are forwarded back to the browser. Otherwise, warn the user by returning the contents of malware_detected.html to the browser.

Proxy server

Your proxy server will need to listen on TCP port 8080. After the browser establishes a connection, you will need to parse the GET and POST messages in order to discover the URL that is being requested. For performance reasons, the browser may make multiple concurrent connections to your proxy server, so your server needs to be multithreaded.

Your proxy should then make a connection to the appropriate server and retrieve the file as given in the URL. When connecting to the server, **you must accept gzip and chunked encodings**. Once the content has been retrieved, you need to reassemble and decompress it as appropriate, then store the resulting data into a file. Decompression can be done in zlib (note: not written by Z. Teo), while chunk reassembly will have to be handled by your own code. This resultant file needs to be stored in its own directory to prevent conflicts with concurrent file retrieval.

Scanning the file

With the raw file data available, your proxy can now check the safety of the content by putting it through a malware scanner. The scanner we are using for this project is ClamAV, which is an open source malware detector. ClamAV comes with an easy-to-use library that can be called directly from your code.

Returning results to the browser

If ClamAV decides that the file is safe, you should serve the file to the browser. Otherwise, you should return an HTML page warning the user of the unsafe content (a template of this HTML is supplied with the project and is named malware_detected.html), preventing the malware from being accessed.

In either case, once the downloaded content has been processed (served to the browser or blocked), you should perform a cleanup by deleting the file and its associated directory.

<u>Turning off the proxy</u>

If you need to turn off proxy connections in Firefox, go under Edit… Preferences… Advanced… Network… Settings… Proxies… and select ‘No proxy’. Don’t forget to turn it on again when you need to test your code.

<u>Other assumptions</u>

Here are some assumptions you can make for the project:

1. You only need to scan ingress files from the web. Files or content uploaded by the user (eg. through the POST command) need not be scanned, compressed, chunked or processed in any way.
2. You do not need to handle SSL connections.
3. Your code does not have to handle byte range requests or deflate encoding.
4. The safety of a given file is dictated entirely by ClamAV, so you do not need to build any additional safety-checking logic for the content.
5. You can assume that HTTP servers will not return malformed headers or content, so any maliciousness will be confined to the file content itself. For example, the HTTP server will not claim to return a compressed file that turns out to be un-decompressible.
6. Files used for testing will not be unreasonably huge (> 100Mb).

<u>Materials provided</u>

We have provided you some well-commented sample programs to get you going for the project:

1. decompress.c contains example code for decompressing a single gzip encoded file. There is a sample gzip file (test.png.gz) for you to test this code on.
2. scan.c contains example code illustrating the steps required to check a file for malware. There are some sample malware in the project directory (sample1/2/3/4.x) for you to run the code against.
3. project.c contains some multithreaded Berkeley sockets code to accept TCP connections. You need to run the project before you run Firefox, or you will not see the incoming TCP connection(s).

4. malware_detected.html contains the template HTML code that you should return to the user when a malware is detected. You can embed the contents of this HTML file directly into your proxy, or fetch it from the disk as required.

Please go over the code to understand how they can be adapted to work for your project. Reading the library and protocol documentation is your responsibility. **We will not answer API or protocol related questions on Piazza**.

Test data

Malware is surprisingly hard to find when you decide to look for it deliberately. There are some websites that allow you to download malware for research purposes. However, doing so may trigger security alerts, especially if you try to access them from campus network connections. In light of this, we have set up a local web server within the VM to host some malware samples. Accessing these samples will not cause alerts because the traffic is internal to the VM.

From within the VM's Firefox browser, type in the URL:

http://localhost

Note that the browser is configured to contact your proxy even for local files. This site contains an HTML page with links to sample malware (these are the same malware files that are located in your project directory). The index HTML file, along with the malware, are located at /var/www/html. You will need superuser privileges to add/remove/modify the contents in this directory.

Bonus credit opportunities

There are two ways to gain bonus credit for this homework:

1. Instead of delivering the static contents of malware_detected.html, modify your proxy such that the HTML file also gives information on the offending URL, such as the filename and malware name.
2. Answer questions for your fellow classmates non-anonymously on Piazza.

Hints

Here are some helpful tips for your project:

1. You may find Wireshark useful for debugging your HTTP code. It may also be helpful to compare the reference browser HTTP transaction against your own to identify bugs in your code.
2. You may want to begin by building a passthrough proxy that parses HTTP requests and retrieves the content from the web, returning the contents to the browser without any

further processing. When you are confident that this works, make your code handle decompression/chunking, as well as the malware scan.

3. Some string functions, eg. strtok(), are not thread-safe. Make sure you account for this when you design your multithreaded code.
4. The libclamav scan engine is slow. It is not a bug in your code.

Submission instructions

Your project archive should contain only the README file, Makefile and project code. Do not include .git, __MAC_OS_X__ directories, test data, malware_detected.html and other irrelevant files. Obviously, your archive should also not contain any malware.

Grading

We will grade your submission by observing its behavior under our own test files. Some additional criteria we will consider are: compilation cleanliness, correctness, performance, memory hygiene and code quality.

Resources

You may find the following resources useful for this homework:

1. Beej's guide to networking (http://beej.us/guide/bgnet/)
2. libclamAV documentation (http://www.clamav.net/documents/installing-clamav)
3. zlib documentation (http://www.zlib.net/manual.html)
4. man pages